stochastic_two_group_model

April 28, 2020

```
In [1]: import numpy as np
        import math
        import matplotlib.pyplot as plt
        import pandas as pd
```

0.1 Stochastic SIR Model - Two Groups

In this notebook, we consider the stochastic SIR model with heterogeneity given by Nandi and Allen (2018). In there paper [1], they describe a heterogeneous population split into two groups. Each group has a susceptible, infective and recovered sub-population with unique transmition and recovery rates. Given in the form of ordinary differential equations (ODEs), the model can be written as

$$\frac{dS_k}{dt} = -\frac{S_k}{N}\sum_{j=1}^{2}\beta_{kj}I_j,$$

$$\frac{dI_k}{dt} = \frac{S_k}{N}\sum_{j=1}^{2}\beta_{kj}I_j - \gamma_k I_k,$$

$$\frac{dI_k}{dt} = \gamma_k I_k, \quad \text{for } k = 1,2.$$

The model currently considered is simplistic, it considers a constant total population $N = N_1 + N_2$ as well constant populations of the two groups $N_j = S_k(t) + I_k(t) + R_k(t)$. In this notebook, we display some results regarding the basic reproductive rate $\mathcal{R}_0$ of the disease for the general case, as well as the simplified cases presented in [1]. First, we generate the next generation matrix; to do so, we consider the states I_1 and I_2 since they are the only states responsible for spreading the disease. By calculating ht next generation matrix FV^{-1} at the disease free equilibrium, we have that

$$FV^{-1} = \begin{bmatrix} \beta_{11}\frac{N_1}{N}\gamma_1^{-1} & \beta_{12}\frac{N_1}{N}\gamma_2^{-1} \\ \beta_{21}\frac{N_2}{N}\gamma_1^{-1} & \beta_{22}\frac{N_2}{N}\gamma_2^{-1} \end{bmatrix}, \quad \text{where the values } \beta_{kj} > 0 \text{ are the parameters defined in the original ODE moc}$$

Basic Reproduction Rate - General We find the $\mathcal{R}_0$ for this disease by calculating the spectral radius of our next generation matrix. The spectral radius the our general next generation matrix FV^{-1} above is

$$\mathcal{R}_0 = \frac{1}{2N}\{\frac{\beta_{11}N_1}{\gamma_1} + \frac{\beta_{22}N_2}{\gamma_2} + \sqrt{(\frac{\beta_{11}N_1}{\gamma_1} + \frac{\beta_{22}N_2}{\gamma_2})^2 - 4\frac{N_1N_2}{\gamma_1\gamma_2}(\beta_{11}\beta_{22} - \beta_{12}\beta_{21})}\} \tag{1}$$

Basic Reproduction Rate - Simplifications In [1], they propose some simplifications to the contact rates B_{jk} by assuming that it is seperable, where the transmission rates depend either on the infectivity group of the host, or the group of the susceptible individual. To model this assumption, we let $\beta_{kj} = \alpha_k \lambda_j$ the product of infective and susceptible parameters. For a group depending in which the transmission of infection depends largely on the infectivity of the host, we have that $\beta_{jk} = \alpha_k \lambda_j = \beta_j^I$. Likewie for a model in which transmission depends on the susceptibilty of the host, we have $\beta_{jk} = \beta_k^S$.

For the above transmission rate simplifications, we obtain basic reproductive rates of similar forms, since the $\beta_{12}\beta_{21}$ term in the square-root of (1) cancels with that of $\beta_{11}\beta_{22}$. The $\mathcal{R}_0$ for both model simplifcations are

$$\text{(Infective dependent transmission)} \quad \mathcal{R}_0 = \frac{\beta_1^I}{\gamma_1}\frac{N_1}{N} + \frac{\beta_2^I}{\gamma_2}\frac{N_2}{N}$$

$$\text{(Susceptible dependent transmission)} \quad \mathcal{R}_0 = \frac{\beta_1^S}{\gamma_1}\frac{N_1}{N} + \frac{\beta_2^S}{\gamma_2}\frac{N_2}{N}$$

References [1] - Nandi, Aadrita and Allen, Linda JS. Stochastic two-group models with transmission dependent on host infectivity or susceptibility. *Journal of biological dynamics*. 2019. Pgs 201 - 224.

0.1.1 Simulations - $\mathcal{R}_0$ = 2.38

Using the parameters presented in the paper by Nandi and Allen [1].

```
In [ ]: def gillespie_algo(t_final, X_0, rates):

            """

            Gillespie's Algorithm implementation for the specifc two-group model given by Nand

            Inputs
            ------------------------------------------------------------------------------------
            t_final - End the algorithm at time t_final

            X = [X[0], X[1], X[2], X[3], X[4], X[5]] where
            X[0] - S_1
            X[1] - S_2
            X[0] - I_1
            X[0] - I_2
            X[0] - R_1
            X[0] - R_2

            and rates = [r_1, r_2, r_3, r_4, r_5, r_6] with
            r_1 = B_11
            r_2 = B_12
            r_3 = B_21
            r_4 = B_22
```

```
r_5 = y_1
r_6 = y_2

Returns
------------------------------------------------------------------------------
t_hist - vector of cumulated waiting times between events
state_hist - array of the values of each state in the model, updated after every e

"""

#import numpy as np
#import math

def calc_a0(X, rates, N, return_all = True):
    a1 = calc_a1(X, rates[0], N)
    a2 = calc_a2(X, rates[1], N)
    a3 = calc_a3(X, rates[2], N)
    a4 = calc_a4(X, rates[3], N)
    a5 = calc_a5(X, rates[4])
    a6 = calc_a6(X, rates[5])
    if return_all:
        return a1+a2+a3+a4+a5+a6, [a1, a2, a3, a4, a5, a6]
    else:
        return a1+a2+a3+a4+a5+a6

# propensity functions

def calc_a1(X, rate, N):
    if X[2] == 0:
        return 0
    else:
        # transmition propensity: - S_1/N*(B_11 * I_1)
        return (X[0]/N) *(rate*X[2])

def calc_a2(X, rate, N):
    if X[3] == 0:
        return 0
    else:
        # transmition propensity: - S_1/N*(B_12 * I_2)
        return (X[0]/N) *(rate*X[3])

def calc_a3(X, rate, N):
    if X[2] == 0:
        return 0
    else:
        # transmition propensity: - S_2/N*(B_21 * I_1)
        return (X[1]/N) *(rate*X[2])
```

```
def calc_a4(X, rate, N):
    if X[3] == 0:
        return 0
    else:
        # transmition propensity: - S_2/N*(B_22 * I_2)
        return (X[1]/N) *(rate*X[3])

def calc_a5(X, rate):
    if X[2] == 0:
        return 0
    else:
        # recovery propensity: - y_1*I_1
        return rate*X[2]

def calc_a6(X, rate):
    if X[3] == 0:
        return 0
    else:
        # recovery propensity: - y_1*I_1
        return rate*X[3]

# changes to state space

def v1():
    # transmition, S_1 + I_1 -> I_1 + I_1
    return np.array((-1, 0, +1, 0, 0, 0))

def v2():
    # transmition, S_1 + I_2 -> I_2 + I_1
    return np.array((-1, 0, +1, 0, 0, 0))

def v3():
    # transmition, S_2 + I_1 -> I_1 + I_2
    return np.array((0, -1, 0, +1, 0, 0))

def v4():
    # transmition, S_2 + I_2 -> I_2 + I_2
    return np.array((0, -1, 0, +1, 0, 0))

def v5():
    # recovery, I_1 -> R_1
    return np.array((0, 0, -1, 0, +1, 0))

def v6():
    # recovery, I_2 -> R_2
    return np.array((0, 0, 0, -1, 0, +1))

N = np.sum(X_0)
```

```
    t = 0
    state_hist = []
    t_hist = []

    prop_fns = {0:v1, 1:v2, 2:v3, 3:v4, 4:v5, 5:v6}

    X = np.asarray(X_0)

    while t <= t_final:
        # calculate a_0
        a_0, a_list = calc_a0(X, rates, N)
        if a_0 == 0:
            break
        # generate a reaction time
        r1 = np.random.rand()
        tau = math.log(1/r1) / (a_0)
        t += tau
        # choose a reaction
        r2 = np.random.rand()
        for idx in range(len(a_list)):
            #print("a_i: " + str(np.sum(a_list[:idx + 1])))
            if r2*a_0 < np.sum(a_list[:idx +1]):
                reaction_idx = idx
                break
            elif idx == (len(a_list) - 1):
                reaction_idx = idx
        # update state vector
        X = X + prop_fns[reaction_idx]()
        state_hist.append(X)
        t_hist.append(t)
    return t_hist, state_hist
```

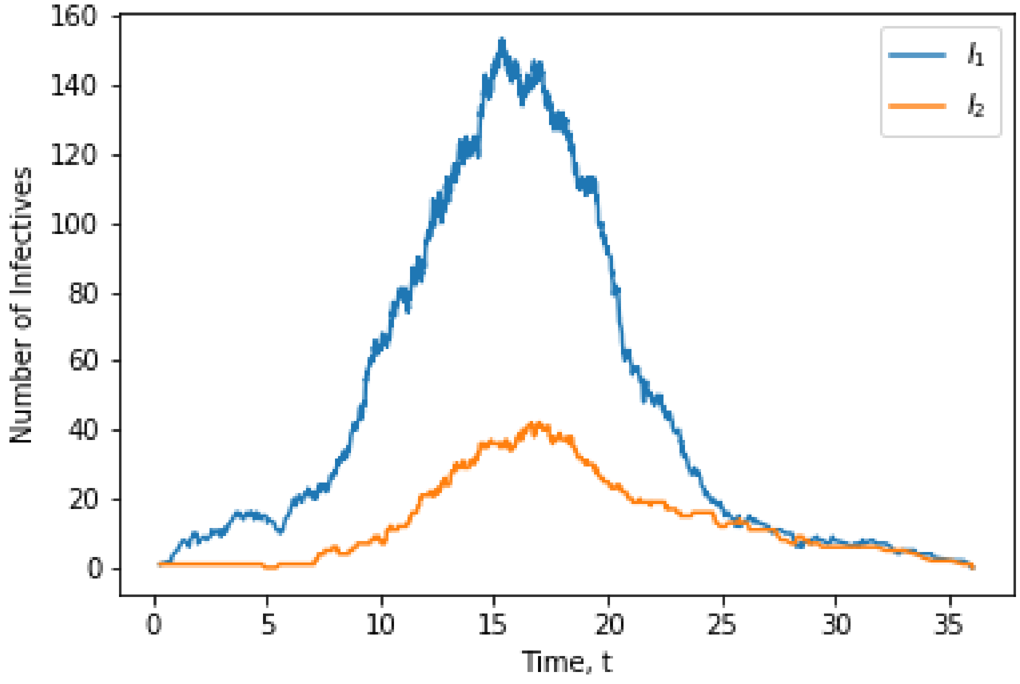
Infectivity dependent transmission

```
In [141]: import time

          start = time.time()
          # using the default parameters in the paper by Nandi and Allen
          time_hist, state_hist = gillespie_algo(100, X_0 = [900, 100, 0, 1, 0, 0], rates = [0
          end = time.time()
          print("Completed in: " +str(end - start) + " seconds")
          print(time_hist[-2])
```

```
Completed in: 0.10866665840148926 seconds
35.98868346123634
```

```
In [142]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [143]: plt.plot(time_hist, I_1)
          plt.plot(time_hist, I_2)
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



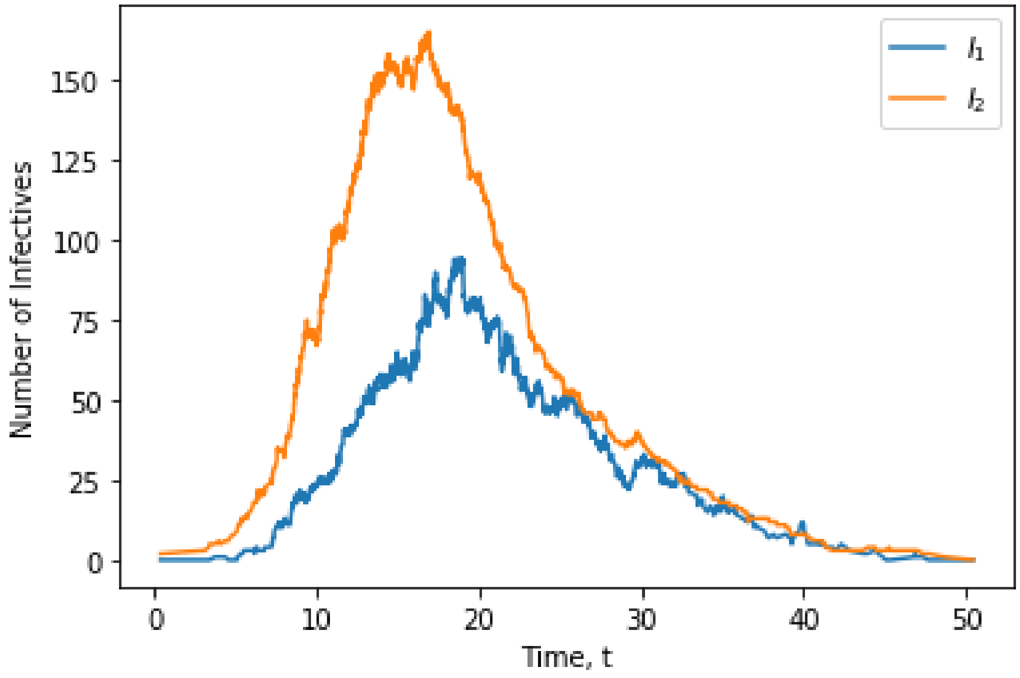
Host susceptibility dependent transmission

```
In [144]: import time

          start = time.time()
          # using the default parameters in the paper by Nandi and Allen
          time_hist, state_hist = gillespie_algo(100, X_0 = [5000, 500, 0, 1, 0, 0], rates = [
          end = time.time()
          print("Completed in: " +str(end - start) + " seconds")
          print(time_hist[-2])
```

```
Completed in: 0.14583992958068848 seconds
48.63968526050156
```

```
In [145]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [146]: plt.plot(time_hist, I_1)
          plt.plot(time_hist, I_2)
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



0.1.2 Simulations - Varying $\mathcal{R}_0$ and transmission dependance

In this section, we extend the model presented in [1] by considering different types of population interactions, along with different $\mathcal{R}_0$ values. We start by considering the infected/susceptible dependent transmission models above, however altering the the parameters so that the $\mathcal{R}_0$ lie closer to 1 and then below one.

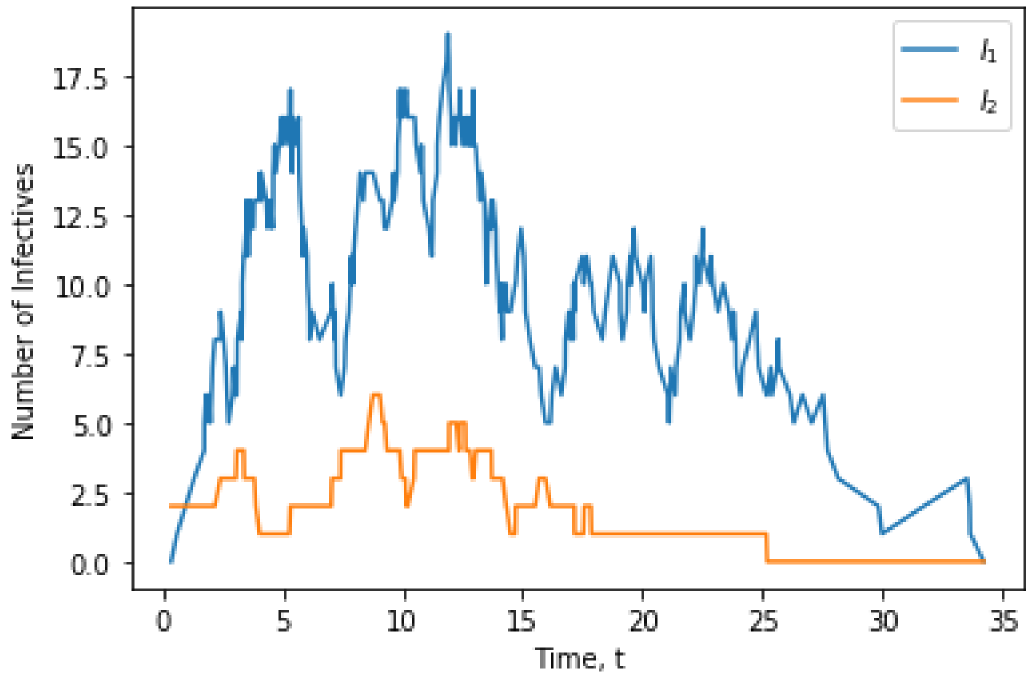
Simulations - $\mathcal{R}_0 = 1.2$, driven by infectivity of host.

```
In [202]: import time
          R_0 = 1.2
          B_1 = 0.2
          B_2 = 2*(R_0 - 2*B_1*0.9)


          start = time.time()
          # using the default parameters in the paper by Nandi and Allen
          time_hist, state_hist = gillespie_algo(100, X_0 = [900, 100, 0, 1, 0, 0], rates = [B
          end = time.time()
          print("Completed in: " +str(end - start) + " seconds")
          print(time_hist[-2])
```

```
Completed in: 0.04284811019897461 seconds
33.65761900350261
```

```
In [203]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [204]: plt.plot(time_hist, I_1, drawstyle = "steps-pre")
          plt.plot(time_hist, I_2, drawstyle = "steps-pre")
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



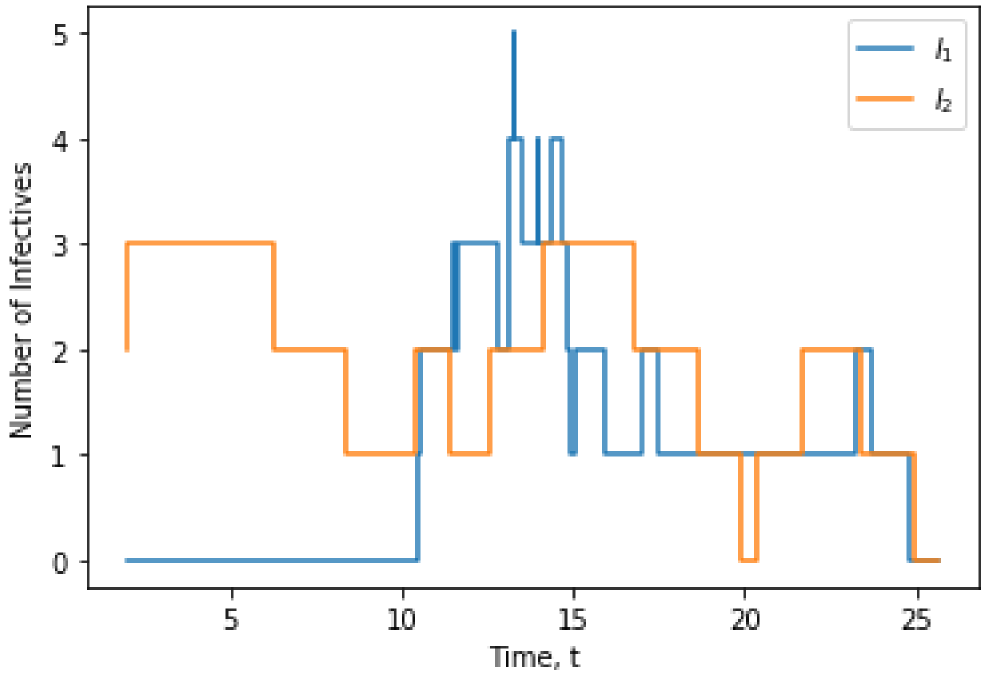
```
In [205]: B_2

Out[205]: 1.6799999999999997
```

Simulations - $\mathcal{R}_0 = 0.85$, driven by infectivity of host - Minor Epidemic

```
In [235]: import time
          R_0 = 0.85
          B_1 = 0.2
          B_2 = 2*(R_0 - 2*B_1*0.9)


          start = time.time()
          # using the default parameters in the paper by Nandi and Allen
          time_hist, state_hist = gillespie_algo(100, X_0 = [900, 100, 0, 1, 0, 0], rates = [B
          end = time.time()
          print("Completed in: " +str(end - start) + " seconds")
          print(time_hist[-2])

Completed in: 0.0029594898223876953 seconds
24.916543550034103


In [236]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
```

```
I_1 = [item[2] for item in state_hist]
I_2 = [item[3] for item in state_hist]
R_1 = [item[4] for item in state_hist]
R_2 = [item[5] for item in state_hist]
```

```
In [237]: plt.plot(time_hist, I_1, drawstyle = "steps-pre")
          plt.plot(time_hist, I_2, drawstyle = "steps-pre")
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



0.1.3 Strong within-group infections

We alter the previously given models to consider a population which is susceptible to a disease. In this section, we consider a population of two groups, where there is strong transmission of the disease within the groups, and relatively weak transmission of the disease between groups (that is the groups do not mix often in comparison to their within group mixing). To model such a scenario, we consider the contact rates β_{jk}; for the within-group contact rates to be stronger, we require that $\beta_{11} >> \beta_{21}$ and $\beta_{22} >> \beta_{12}$.

Simulation: $\mathcal{R}_0 = 2.5$

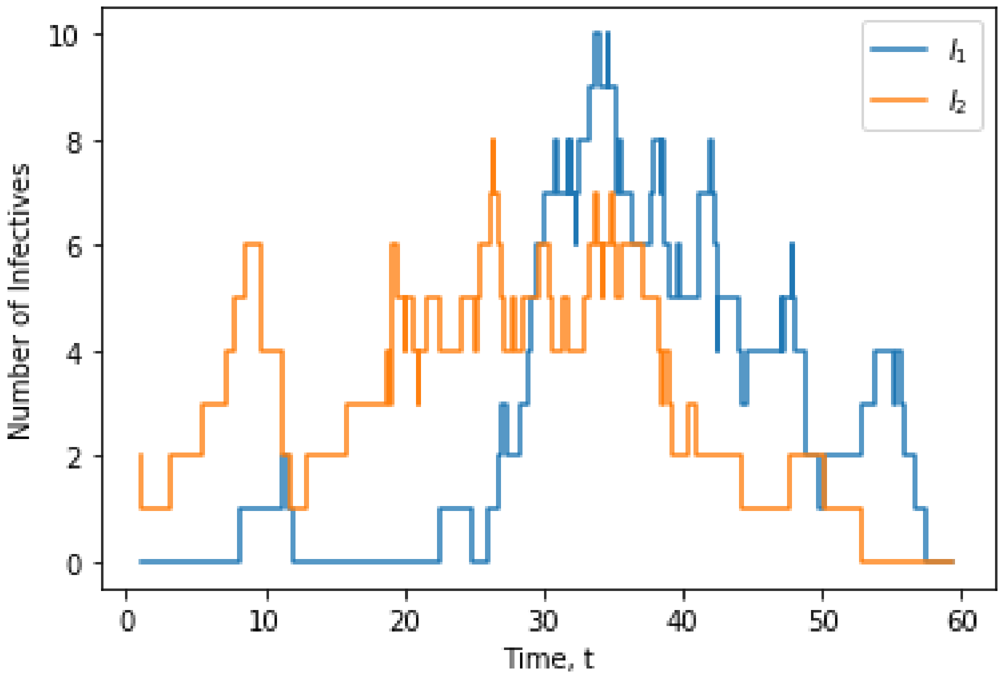
```
In [310]: import time
          # R_0 = 2.3
```

```
        # B_1 = 0.2
        # B_2 = 2*(R_0 - 2*B_1*0.9)

        B_11 = 0.6
        B_12 = 0.05
        B_21 = 0.05
        B_22 = 0.4
        N_1 = 500
        N_2 = 500
        N = N_1 + N_2
        y_1 = 0.2
        y_2 = 0.2

        print("R_0 = " + str(  (1/(2*N)) * ((B_11*N_1)/(y_1) + (B_22*N_2)/(y_2) + math.sqrt(

        start = time.time()
        # using the default parameters in the paper by Nandi and Allen
        time_hist, state_hist = gillespie_algo(100, X_0 = [500, 500, 0, 1, 0, 0], rates = [B
        end = time.time()
        print("Completed in: " +str(end - start) + " seconds")
        print(time_hist[-2])
```

```
R_0 = 2.4990496387253787
Completed in: 0.005991935729980469 seconds
57.441785514546304
```

```
In [311]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [312]: plt.plot(time_hist, I_1, drawstyle = "steps-pre")
          plt.plot(time_hist, I_2, drawstyle = "steps-pre")
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



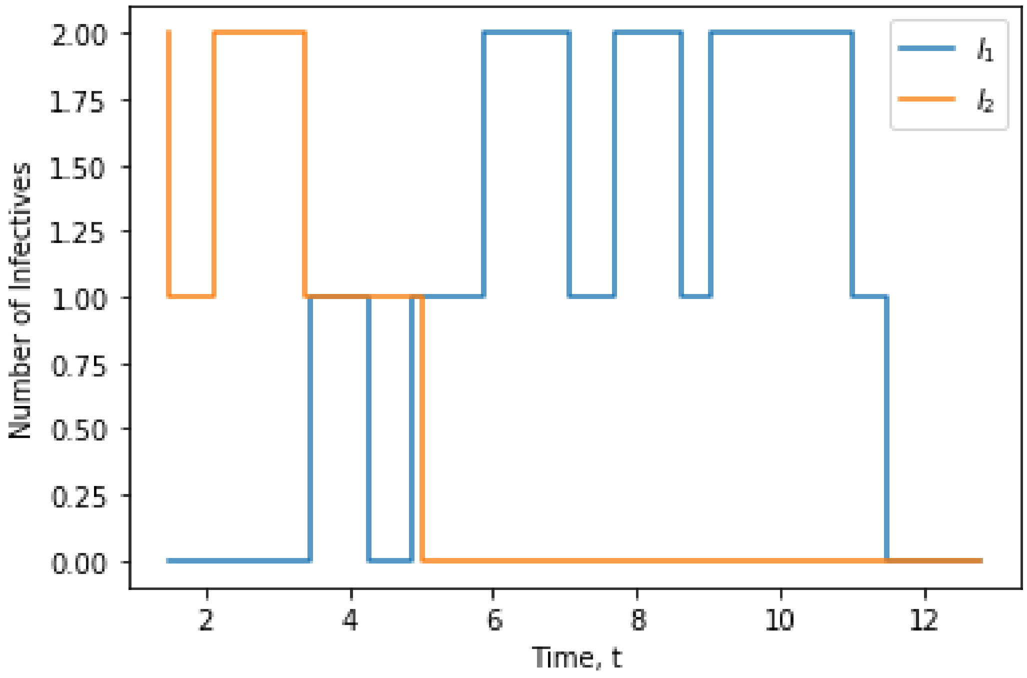
Simulation: $\mathcal{R}_0 = 1.2$

```
In [372]: import time
          # R_0 = 2.3
          # B_1 = 0.2
          # B_2 = 2*(R_0 - 2*B_1*0.9)

          B_11 = 0.8
          B_12 = 0.05
          B_21 = 0.05
          B_22 = 0.7
          N_1 = 500
          N_2 = 500
          N = N_1 + N_2
          y_1 = 0.5
          y_2 = 0.5

          print("R_0 = " + str(  (1/(2*N)) * ((B_11*N_1)/(y_1) + (B_22*N_2)/(y_2) + math.sqrt(

          start = time.time()
          # using the default parameters in the paper by Nandi and Allen
          time_hist, state_hist = gillespie_algo(100, X_0 = [500, 500, 0, 1, 0, 0], rates = [B
          end = time.time()
          print("Completed in: " +str(end - start) + " seconds")
          print("Total waitig time: " + str(time_hist[-2]))
```

```
R_0 = 1.4763995112883819
Completed in: 0.0009970664978027344 seconds
Total waitig time: 11.485203621780744
```

```
In [373]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [374]: plt.plot(time_hist, I_1, drawstyle = "steps-pre")
          plt.plot(time_hist, I_2, drawstyle = "steps-pre")
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



Simulation: $\mathcal{R}_0 = 0.9$

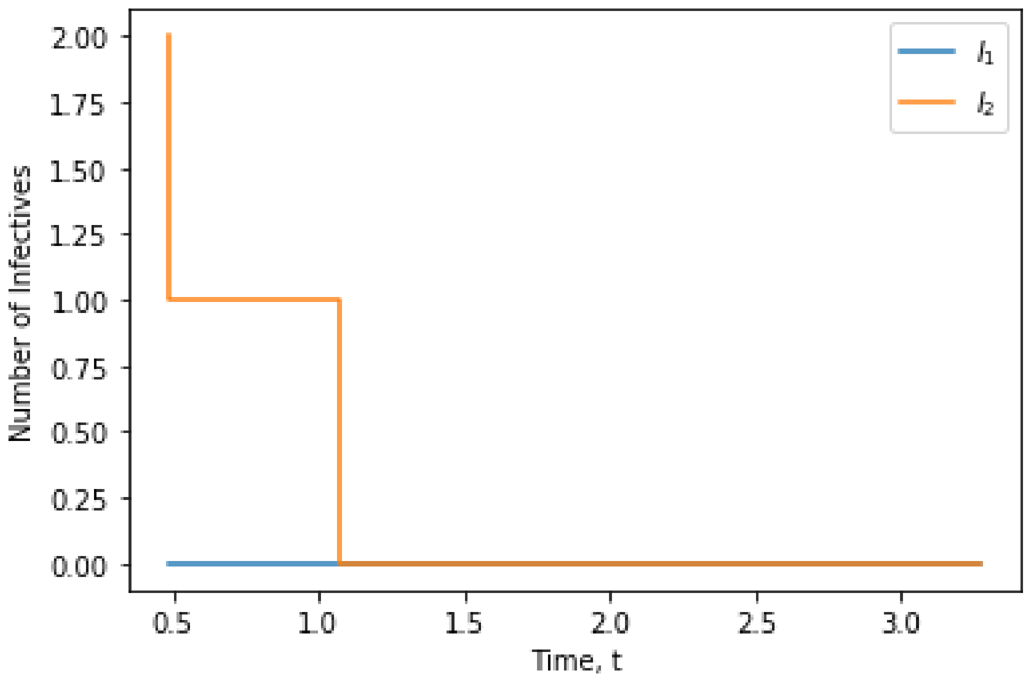
```
In [376]: import time
          # R_0 = 2.3
          # B_1 = 0.2
          # B_2 = 2*(R_0 - 2*B_1*0.9)
```

```
        B_11 = 0.5
        B_12 = 0.05
        B_21 = 0.05
        B_22 = 0.45
        N_1 = 500
        N_2 = 500
        N = N_1 + N_2
        y_1 = 0.5
        y_2 = 0.5

        print("R_0 = " + str(  (1/(2*N)) * ((B_11*N_1)/(y_1) + (B_22*N_2)/(y_2) + math.sqrt(

        start = time.time()
        # using the default parameters in the paper by Nandi and Allen
        time_hist, state_hist = gillespie_algo(100, X_0 = [500, 500, 0, 1, 0, 0], rates = [B_
        end = time.time()
        print("Completed in: " +str(end - start) + " seconds")
        print("Total waitig time: " + str(time_hist[-2]))
```

```
R_0 = 0.9351290579826491
Completed in: 0.0 seconds
Total waitig time: 1.072494798380558
```

```
In [377]: S_1 = [item[0] for item in state_hist]
          S_2 = [item[1] for item in state_hist]
          I_1 = [item[2] for item in state_hist]
          I_2 = [item[3] for item in state_hist]
          R_1 = [item[4] for item in state_hist]
          R_2 = [item[5] for item in state_hist]
```

```
In [378]: plt.plot(time_hist, I_1, drawstyle = "steps-pre")
          plt.plot(time_hist, I_2, drawstyle = "steps-pre")
          plt.legend(["$I_1$", "$I_2$"])
          plt.ylabel("Number of Infectives")
          plt.xlabel("Time, t")
          plt.show()
```



2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Number of Infectives
0.5
1.0
1.5
2.0
2.5
3.0
Time, t
I_1
I_2

```
In [ ]:
```